

Venn Map<<<

Chapter 1: Number Representation and Arithmetic Circuit

Unsigned Number (numbers that are positive only)

Decimal integer: n-tuple comprising n decimal digits $D = d_{n-1}d_{n-2}...d_1d_0$

Value of D $V(D) = d_{n-1} \times 10^{n-1} + d_{n-2} \times 10^{n-2} + ... + d_1 \times 10^1 + d_0 \times 10^0$ (positional number representation) base/radix-10 number

Binary Number: 0, 1

Each binary digits is called a **bit**. A group of 4 bits is called a **nibble**, A group of 8 bits is called a **byte**.

Value of B $V(B) = b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + ... + b_1 \times 2^1 + b_0 \times 2^0 = \sum b_i \times 2^i$

Conversion of numbers from decimal to binary:

① successively-divide example:

② construct the number by using powers of 2. (think about the largest powers of 2 fit the number)

③ For larger number: $(25600)_{10} = 2^8 \times 100 = 2^8 \times (64 + 32 + 4) = 2^8 \times (2^6 + 2^5 + 2^2) = 2^{14} + 2^{13} + 2^{10} = (11001\ 00000\ 00000)_2$

Note: power of 2 indicates 1s in particular place.

$(2^0 \sim 1)(2^1 \sim 2)(2^2 \sim 4)(2^3 \sim 8)(2^4 \sim 16)(2^5 \sim 32)(2^6 \sim 64)(2^7 \sim 128)(2^8 \sim 256)(2^9 \sim 512)(2^{10} \sim 1024)(2^{11} \sim 2048)(2^{12} \sim 4096)(2^{13} \sim 8192)$

Conversion of numbers from binary to decimal:

example: $(1111)_2 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 15$

minimum number of bits need to represent a decimal number: $2^{n-1} \leq \text{decimal number} < 2^n$ need n bits.

Using n bits allows representation of integers in the range of 0 to $2^n - 1$.

least-significant bit & most-significant bit

for binary number $A = b_{n-1}b_{n-2}...b_1b_0$ $2A = A \ll 1 = b_{n-1}b_{n-2}...b_1b_00$

Octal Number: 0 1 2 3 4 5 6 7

Hexadecimal Number: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

convert binary to hexadecimal: $(1\ 1001\ 0110\ 1100\ 0101)_2 = (1\ 9\ 6\ C\ 5)_{16}$

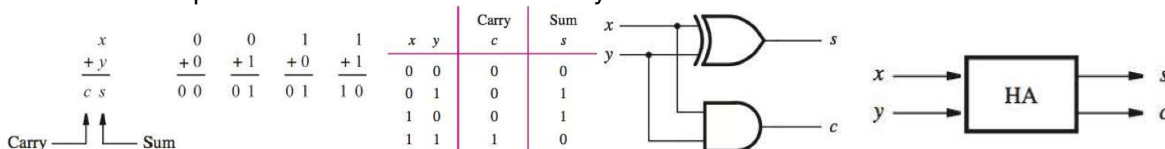
3 bits as a group

4 bits as a group

In General: $K = k_{n-1}k_{n-2}...k_1k_0$ Value of K $= k_{n-1} \times R^{n-1} + k_{n-2} \times R^{n-2} + ... + k_1 \times R^1 + k_0 \times R^0 = \sum k_i \times R^i$ (from 0 to n-1) R: radix

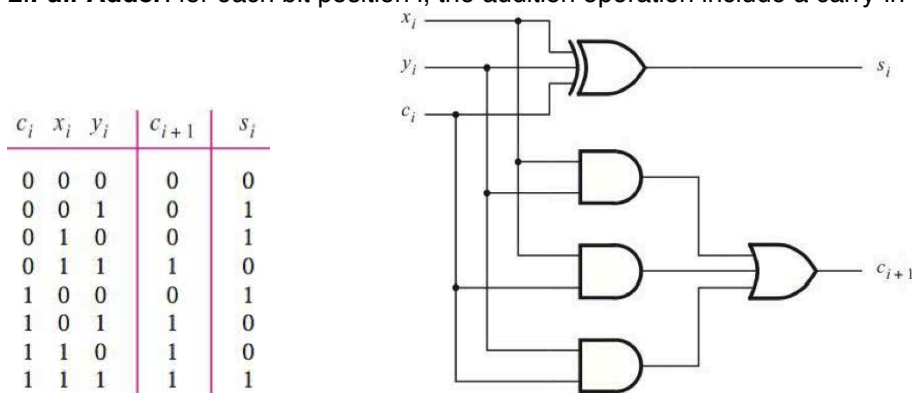
Addition of Unsigned Binary Numbers: Carry in/Carry out

1. **Half-Adder:** implementation of the addition of only 2 bits



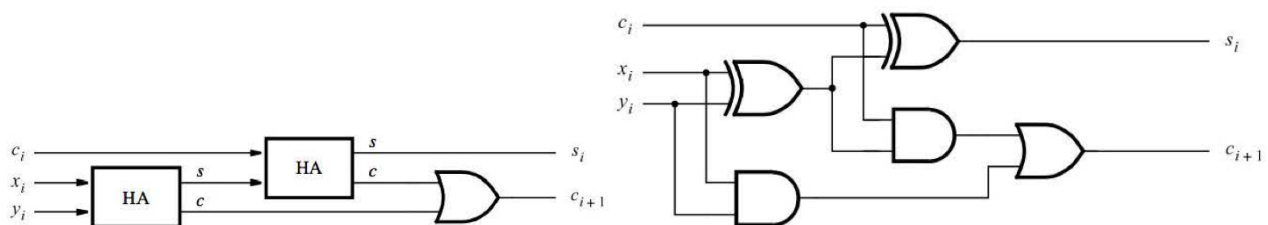
$c = xy$; $s = x \oplus y$;

2. **Full-Adder:** for each bit position i, the addition operation include a carry-in from bit position i-1.



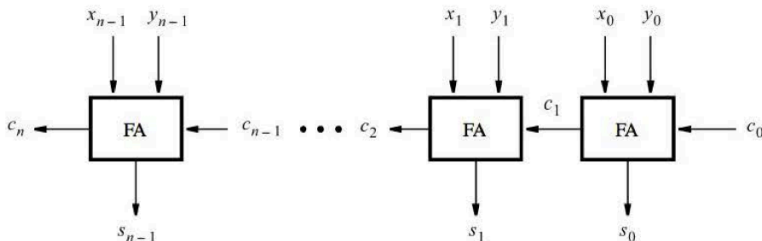
$c_{i+1} = xy + c(x+y)$; $s_i = c \oplus x \oplus y$;

Decomposed Full Adder: constructed by two half adder



3. Ripple-Carry Adder:

carry signals “ripple” through the full-adder



```
module halfadder(a,b,s,Cout);
  input a,b;
  output s,Cout;

  assign s=a^b;
  assign Cout=a&b;
endmodule
```

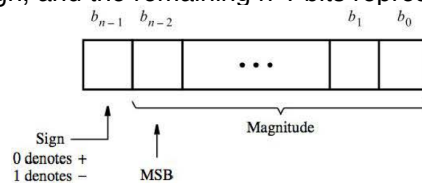
```
module module fulladder(Cin, x,y,s,Cout);
  input Cin, x, y;
  output s,Cout;

  assign s=x^y^Cin;
  assign Cout=(x&y) | (Cin & (x+y));
endmodule
```

```
module ripplecarry(input [3:0]X,Y, input Cin, output [3:0] S,Cout);
  wire [2:0]C; //4 bits X+ 4 bits Y
  fulladder u1(Cin, X[0],Y[0], S[0],C[0]);
  fulladder u2(C[0], X[1],Y[1], S[1],C[1]);
  fulladder u3(C[1], X[2],Y[2], S[2],C[2]);
  fulladder u4(C[2], X[3],Y[3], S[3],Cout);
endmodule
```

Signed Number (numbers that can be negative)

Sign of a number is denoted by left-most bit. (positive 0, negative 1)
the left-most bit represents the sign, and the remaining n-1 bits represent the magnitude.



MSB is in bit position b_{n-2}

Signed number

Positive number Representation: positional number representation

Negative number Representation:

① **Sign-and-Magnitude:** (not suited for use in computer) E.g. +5=0101 -5=1101

② **1's complement:** $(N_1)_{10} = (2^n - 1) - (P)_{10}$ n-bit negative num N & its equivalent positive num P
E.g. n=4 then -5=(2⁴-1)-5=(10)₁₀ =(1111)-0101=(1010)₂

Simply complement all bits including sign bit of the positive number.

③ **2's complement:** $(N_2)_{10} = 2^n - (P)_{10}$ $(N_2)_{10} = (N_1)_{10} + 1$

E.g. n=4 then -5=2⁴-5=(11)₁₀=(10000)-0101=(1011)₂

(most efficient method for performing addition and subtraction operations)

Finding 2's Complements: >complement all bits of $(P)_2$ and then add 1.

> (start from left) copy all 0 bits and first 1 bit of $(P)_2$; then complement the rest bits to get $(N_2)_2$.

2's complement: positive equivalent value: +800=8(64+32+4)=2³×(2⁶+2⁵+2²)=2⁹+2⁸+2⁵

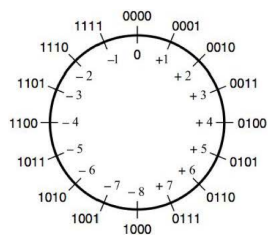
+800: 0000, 00 1 1, 0 0 1 0, 0 0 0 0 (16-bits) -800: 1111, 1100, 1110, 0000

$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0111	+7	+7	+7
0110	+6	+6	+6
0101	+5	+5	+5
0100	+4	+4	+4
0011	+3	+3	+3
0010	+2	+2	+2
0001	+1	+1	+1
0000	+0	+0	+0
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

$$\text{Value of } B: V(B) = (-b_{n-1} \times 2^{n-1}) + b_{n-2} \times 2^{n-2} + \dots + b_1 \times 2^1 + b_0 \times 2^0$$

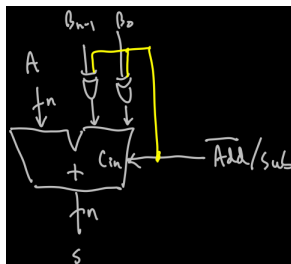
Largest negative number $100..00 = -2^{n-1}$ Largest positive number $011..11 = 2^{n-1}-1$

Note: there are distinct representations for +0 and -0 in sign-and-magnitude & 1's-complement systems, but 2's-complement system has only one representation for 0.

2's Complement Subtraction		2's Complement Addition	
$(+5) + (+2) = +7$ $\begin{array}{r} 0101 \\ + 0010 \\ \hline 0111 \end{array}$	$(-5) + (+2) = -3$ $\begin{array}{r} 1011 \\ + 0010 \\ \hline 1101 \end{array}$	$(+5) - (+2) = +3$ $\begin{array}{r} 0101 \\ - 0010 \\ \hline 0011 \end{array}$	$(-5) - (+2) = -7$ $\begin{array}{r} 1011 \\ - 0010 \\ \hline 1001 \end{array}$
$(+5) + (-2) = +3$ ignore carry-out $\begin{array}{r} 0101 \\ + 1110 \\ \hline 1001 \end{array}$	$(-5) + (-2) = -7$ $\begin{array}{r} 1011 \\ + 1110 \\ \hline 1001 \end{array}$	$(+5) - (-2) = +7$ $\begin{array}{r} 0101 \\ - 1110 \\ \hline 0111 \end{array}$	$(-5) - (-2) = -3$ $\begin{array}{r} 1011 \\ - 1110 \\ \hline 1101 \end{array}$
1's Complement Addition		Graphical Representation for 2's complement	
$(+5) + (+2) = +7$ $\begin{array}{r} 0101 \\ + 0010 \\ \hline 0111 \end{array}$	$(-5) + (+2) = -3$ $\begin{array}{r} 1010 \\ + 0010 \\ \hline 1100 \end{array}$	 <p>modulo 2^n (16 in this case) circle</p> <p>Addition/Subtraction operation is performed by stepping in the clockwise/counterclockwise direction by the magnitude of the number to be added/subtracted.</p>	
$(+5) + (-2) = +3$ $\begin{array}{r} 0101 \\ + 1101 \\ \hline 10010 \\ \text{Carry } 1 \rightarrow \\ \hline 0011 \end{array}$	$(-5) + (-2) = -7$ $\begin{array}{r} 1010 \\ + 1101 \\ \hline 10111 \\ \text{Carry } 1 \rightarrow \\ \hline 1000 \end{array}$		

Adder & Subtractor Unit: $X \pm Y$ $\overline{\text{Add}} / \text{Sub} = 0(\text{add})$ $\overline{\text{Add}} / \text{Sub} = 1(\text{subtract})$

Note: XOR A/S with Y, then add to X and use A/S as C_{in}



Radix-Complement Scheme

Arithmetic Overflow: actual result of an arithmetic operation is outside the representable range.

Unsigned Number: a carry-out of 1 from the most significant bit position indicates that an overflow has occurred.

Signed Number: overflow may occur only if both summands have the same sign



Fixed-Point number / Floating-Point number

Chapter 2: Introduction to Logic Circuits

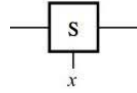
Digit Circuit: each signal value is represented by a digit.

Analog Circuit: signals take on a continuous range of values between some minimum and maximum levels.

Variable and Functions

(Binary) Switch open if $x=0$  Switch closed if $x=1$ 

graphical symbol



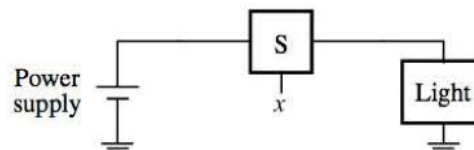
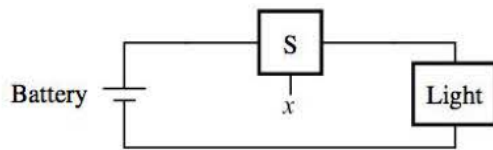
transistor

Moore's Law: the number of [transistors](#) on [integrated circuits](#) doubles approximately every two years.

Connection to a lightbulb:

Light: On if $L=1$; Off if $L=0$.

$L(x)=x$



① simple connection

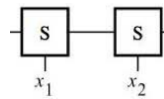
② a ground connection as return path

Logical expression/function: output as a function of input(s)

in electronic circuit: switch implemented as transistor, light as light-emitting diode(LED).

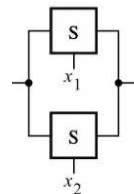
Logic Operations:

① AND (series connection) $L(x,y)=x \cdot y=xy$

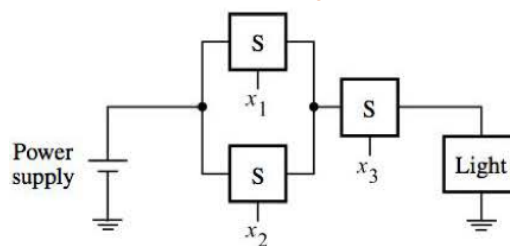


in Verilog: &

② OR (parallel connection) $L(x,y)=x+y$



in Verilog: |



in Verilog: ~, !

③ NOT (inverse/complement) $L(x)=\bar{x}$

$\bar{x} = x'$ (apostrophe) = !x (exclamation mark) = ~ x (tilde character)

Truth Table

x_1	x_2	$x_1 \cdot x_2$	$x_1 + x_2$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

AND

OR

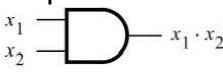
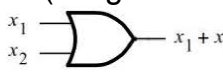
left column: all possible combinations of logic values(=valuation) of input variables

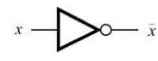
Logic Gates and Networks

A logic gate has one or more inputs and one output that is a function of its inputs.

Circuit Diagram/Schematic: describe a logic circuit (consist of graphical symbols representing logic circuit)

A large circuit is implemented by a network of gates. (a logic network/circuit)

① AND gates:  ② OR gates: 

③ NOT gates: 

Analysis process: determine the function performed by the network for an existing logic network

Synthesis process: design a new network that implements a desired functional behavior/ generate a circuit that realizes desired functional behavior.

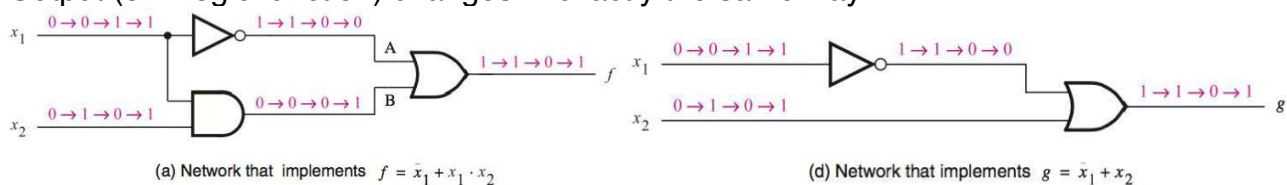
Functional Behavior of Logic network/circuits: Truth table/Timing Diagram(不考虑delay)

Timing Diagram: (waveforms equivalent to Truth Table)

time runs from left to right, each input valuation held for some fixed period, figure shows waveforms for inputs & outputs of the network and the internal signals.

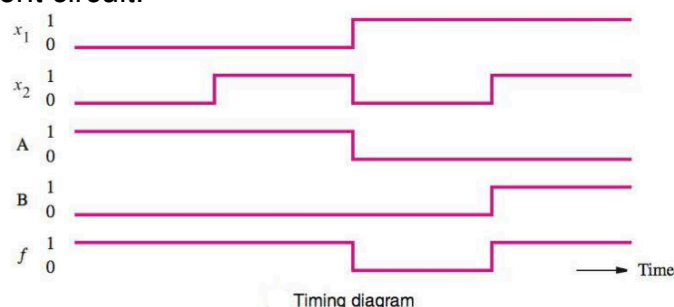
Functionally Equivalent Network:

Output (of 2 logic function) changes in exactly the same way.



f and g are functionally equivalent circuit.

x_1	x_2	$f(x_1, x_2)$	A	B
0	0	1	1	0
0	1	1	1	0
1	0	0	0	0
1	1	1	0	1



Boolean Algebra

Axiom	Dual Form	Duality:
① $0 \cdot 0 = 0$	$1 + 1 = 1$	Duality: Given a logic expression, its dual is obtained by replacing all + operators with \cdot operators and by replacing all 0s with 1s. Note: the dual of any true statement(axiom/theorem) in Boolean Algebra is also true.
② $1 \cdot 1 = 1$	$0 + 0 = 0$	
③ $0 \cdot 1 = 1 \cdot 0 = 0$	$0 + 1 = 1 + 0 = 1$	
④ If $x=0$, then $\bar{x}=1$	If $x=1$, then $\bar{x}=0$	

Theorem	SingleVari	Theorem	Two or More Vari	
⑤ $x \cdot 0=0$	$x+1=1$	⑩commutative	$x \cdot y=y \cdot x$	$x+y=y+x$
⑥ $x \cdot 1=x$	$x+0=x$	⑪associative	$x \cdot (y \cdot z)=(x \cdot y) \cdot z$	$x+(y+z)=(x+y)+z$
⑦ $x \cdot x=x$	$x+x=x$	⑫distributive	$x \cdot (y+z)=x \cdot y+x \cdot z$	$x+(y \cdot z)=(x+y) \cdot (x+z)$
⑧ $x \cdot \bar{x}=0$	$x+\bar{x}=1$	⑬absorption/covering	$x+x \cdot y=x$	$x \cdot (x+y)=x$
		⑭combining	$x \cdot y+x \cdot \bar{y}=x$	$(x+y) \cdot (x+\bar{y})=x$
		⑮DeMorgan 's	$\overline{x \cdot y}=\bar{x}+\bar{y}$ <small>(prove by truth table)</small>	$\overline{x+y}=\bar{x} \cdot \bar{y}$
		⑯Consensus/absorption	$x+\bar{x} \cdot y=x+y$ Proof: $x=0 \dots x=1 \dots$	$x \cdot (\bar{x}+y)=x \cdot y$
⑰ $(x+y) \cdot (y+z) \cdot (\bar{x}+z)=(x+y) \cdot (\bar{x}+z)$			$xy+yz+\bar{x}z=xy+\bar{x}z$ Proof: $xy+(x+x')yz+x'z=xy+xyz+x'yz+x'z=xy(1+z)+x'(y+1)z=xy+x'z$	

Proof: i) by Truth Table (perfect induction) ii) by Axioms

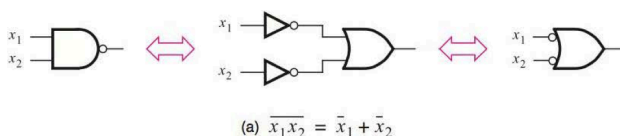
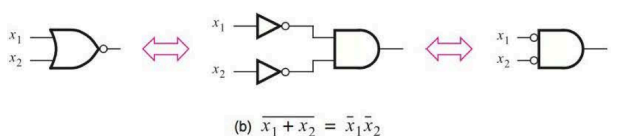
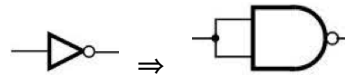
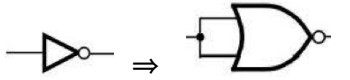
Procedure of Operations: convention-- operations in a logic expression must be performed in the order NOT, AND, and then OR. (Parentheses can be used to indicate the order.)

Synthesis using AND, OR, NOT Gates

canonical vs minimal-cost realization

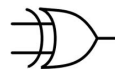
Sum-Of-Products	Product-Of-Sums
Minterm: for a function of n variables, a product term in which each of the n variable appears once (variable either in uncomplemented or complemented form) For a given row of the truth table, the minterm is formed by including x_i if $x_i=1$ and \bar{x}_i if $x_i=0$.	Maxterm: for a function of n variables, a sum term in which each of the n variable appears once For a given row of the truth table, the maxterm is formed by including x_i if $x_i=0$ and \bar{x}_i if $x_i=1$.
Sum-Of-Products: a logic expression consisting of product(AND) terms that are summed(ORed)	Product-Of-Sums:
Canonical SOP: lower-cost functionally equivalent SOP Any function f can be represented by a sum of minterms that correspond to the rows in the truth table for which $f=1$. A function can be represented by a sum of minterms where each minterm is ANDed with the value of f for the corresponding valuation of input variables.	Canonical POS: Any function f can be synthesized by taking maxterm for each row in the truth table for which $f=0$ and forming a product of these maxterms.
Σ : logical sum operation	Π : logical product operation
Example: $f(x,y,z) = \Sigma m(0,1,2,3) = \Pi M(4,5,6,7)$	

Conversion SOP & POS: DeMorgan's Theorem, Distributive, $(x+y)(x+y')=x$

NAND	NOR
DeMorgan's $\overline{x \cdot y} = \bar{x} + \bar{y}$ POS minterm	DeMorgan's $\overline{x + y} = \bar{x} \cdot \bar{y}$ SOP maxterm
AND-OR \Rightarrow NAND-NAND networks  (a) $\overline{x_1 x_2} = \bar{x}_1 + \bar{x}_2$	OR-AND \Rightarrow NOR-NOR networks  (b) $\overline{x_1 + x_2} = \bar{x}_1 \bar{x}_2$
NOT \Rightarrow NAND 	NOT \Rightarrow NOR 

NAND and NOR operation are functionally complete.

XOR gate: exclusive OR $L=x \oplus y$ in Verilog: ^



XOR is "odd function"-- number of inputs are 1 is odd then XOR of those inputs is 1.

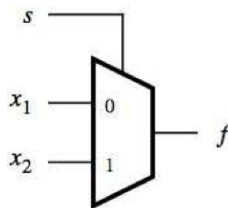
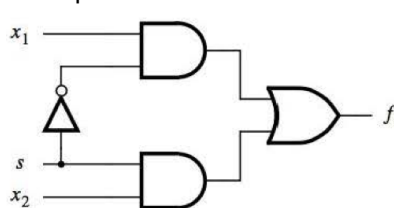
$$x \oplus y = \bar{x}y + x\bar{y} \quad \overline{x \oplus y} = x \cdot y + \bar{x} \cdot \bar{y}$$

Multiplexer (select signal)

A circuit that generates an output that exactly reflects the state of one of a number of data inputs, based on the value of one or more selection control inputs.

A multiplexer circuit "multiplexes" input signals onto a single output.

Example:



$$f(s, x_1, x_2) = \bar{s}x_1 + sx_2 \quad (\text{select signal } s)$$

x-to-y N bits Multiplexer: x(# of inputs) y(# of outputs) N(bit of input/output) need $\log_2 x$ selection signals.

Adder

Carry Function: (Carry in/out)

Majority Function: it is 1 if a majority of its inputs are 1.

A x-variable-input logic function is equal to 1 if no less than $x/2$ of its variables are equal to 1.

Karnaugh Map

Summary of 4-Variable K map:

Group of 2-- 2^1 blocks=3 vari

Group of 4--one row/column=2 vari 2^2 blocks=2 vari

Group of 8-- 2^3 blocks=1 vari

		0	0	0	1	1	1	1	0
0	0	m_0	m_4	\bar{m}_1	\bar{m}_5	m_8	m_{12}	\bar{m}_9	\bar{m}_{13}
0	1	m_1	m_5	\bar{m}_4	\bar{m}_0	m_9	m_{13}	\bar{m}_{12}	\bar{m}_8
1	1	m_3	m_7	\bar{m}_3	\bar{m}_7	m_{11}	m_{15}	\bar{m}_{14}	\bar{m}_{10}
1	0	m_2	m_6	\bar{m}_2	\bar{m}_6	m_{10}	m_{14}	\bar{m}_{15}	\bar{m}_{11}

***Don't-care Minterms:** specific valuations of the inputs to a circuit won't occur in practice, or if they do occur, we don't care what output value is produced by our function.

D(# of minterms)

Minimization Process (derive a minimum-cost implementation)

***Literal:** Each appearance of a variable (uncomplemented/complemented)

***Implicant:** a product term that indicates the input valuation(s) for which a given function is equal to 1. (Most Basic Implicants: minterm)

***Prime Implicant:** An implicant cannot be combined into another implicant that has fewer literals/ impossible to delete any literal in a prime implicant and still have a valid implicant. (largest group of 1's in K-map)

***Cover**(of a function): A collection of implicants that account for all valuations for which a given function is equal to 1.

Note: a cover consisting of prime implicants leads to the lowest-cost implementation.

***Essential Prime Implicant:** A prime implicant that covers at least one minterm exclusively. (no other PI covers this minterms)

***Cost:** = inputs+gates(except NOT gates)

number of inputs: including internal inputs

Example: product term $x y z$ 3 literals

of different minimum-cost POS/SOP expression for f : 一定include essential PI

minimization for SOP	minimization for POS
minterms $f=1$	maxterms $f=0$
1. Generate all prime implicants for the given function f 2. Find the set of Essential Prime Implicant 3. If the EPI covers all valuations for which $f=1$ (0 for POS), then this set is the desired cover of f . Otherwise, determine the nonessential prime implicants that should be added to form a complete minimum-cost cover.	
SOP & POS implementations of a given function may not entail the same cost. SOP \rightarrow POS: complement (DeMorgan's Theorem) example: simplest SOP: $f' = ab' + ac'$ POS $f = f'' = (ab' + ac')' = (ab')'(ac')' = (a' + b)(a' + c)$	

programmable-logic devices(PLDs) predefined basic structure

for large functions, there may exist many possibilities, and some heuristic approach. (an approach that considers only a subset of possibilities but gives good results most of the time)

Incompletely Specified Function

Incompletely Specified Function: a function contains don't-care condition(s).

don't-cares denoted by d in K-map. and as the set of D in function f .

Multiple-Output Circuits

Chapter 3: Sequential Circuit

Combinational circuit: output depends on present value of inputs

Sequential circuit: output depends on present and previous/past stored value of inputs (storage element stores value of logic signals, content of storage elements represent state of circuit)

① Moore type: outputs depend only on the state of circuit

② Mealy type: outputs depend on the state and primary inputs

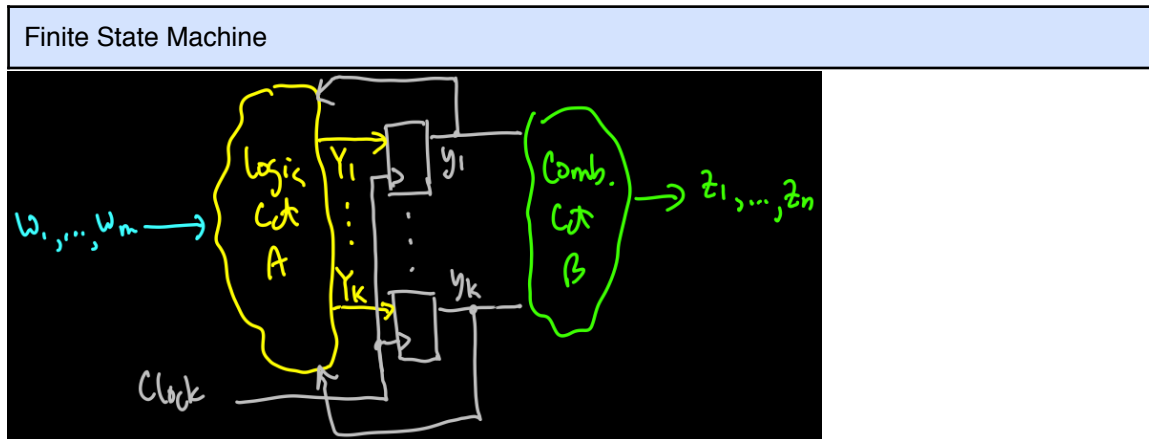
Storage Elements: FFs

Synchronous Sequential Circuit(finite state machine) : a clock signal is used to control the operation of a sequential circuit; (combinational circuit+ edge-triggered Flip-Flops)

Asynchronous sequential circuit: no clock signal is used.

state: values of outputs of FFs. active clock edge>> change in state

sequence detector:



Step:

①state diagram:

②state table: present state & next state

③state assignment:

> minimum # of FF, codes

> one-hot encoding

④choose Flip-Flops and

⑤state assigned table: state vs code Y is output y is input

All FSMs are coded in the same way

blocks of code ① state table ②Flip-Flops ③FSM output

Fully-Encoded State Machine (use less FFs)

```
module FSM(input w,Clock,Resetn, output z);
    reg [2:1] y,Y; //y input,Y output
    parameter A=2'b00, B=2'b01,C=2'b10,D=2'b11;
    //state table
    always @ (w,y)
    case (y)
    A: if(w) Y=B;   else Y=A;
    B: if(w) Y=B;   else Y=C;
    C: if(w) Y=D;   else Y=A;
    D: if(w) Y=B;   else Y=C;
    endcase
    //state FFs
    always @ (posedge Clock)
        if (Resetn==0) y<=A;
        else y<=Y; //output as next input
    //FSM outputs
    ①assign z= (y==D) ;    ②assign z= (y==D)? 1:0;
    ③always @ (*) if (y==D) z=1'b1; else z=1'b0;
endmodule
```

1-Hot Encoded State Machine

```
module FSM(input w,Clock,Resetn, output z);
    reg [4:1] y,Y; //y:present state Y:next state
    parameter A=4'b0001, B=4'b0010,C=4'b0100,D=4'b1000;
    //state table
    //state (output) reached by which state (input)
    assign Y[1]= (~w&y[1])|(~w&y[3]);
    assign Y[2]= (w&y[1])|(w&y[2])|(w&y[4]);
    assign Y[3]= (~w&y[2])|(~w&y[4]);
    assign Y[4]= w&y[3];

    //state FFs
    always @ (posedge Clock)
        if (Resetn==0) y<=A;
        else y<=Y; //output as next input

    //FSM outputs
    assign z= y[4];
endmodule
```

case statement for HEX

```

module seg7BCDdecoder(input [3:0]NUM,output reg [0:6]HEX0); //input is BCD digits //HEX, on=0 off=1
  always @(*)
    case (NUM)
4'b0000:  HEX0=7'b0000001;      #0 // if declare [6:0]HEX0 then =7'b1000000;
4'b0001:  HEX0=7'b1001111;      #1      4'b0010:  HEX0=7'b0010010;      #2      4'b0011:  HEX0=7'b0000110;      #3
4'b0100:  HEX0=7'b1001100;      #4      4'b0101:  HEX0=7'b0100100;      #5      4'b0110:  HEX0=7'b0100000;      #6
4'b0111:  HEX0=7'b0001111;      #7      4'b1000:  HEX0=7'b0000000;      #8      4'b1001:  HEX0=7'b0001100;      #9
default:  HEX0=7'bxxxxxxx;      //dont-cares
    endcase
endmodule

```

if we omit default clause, value of HEX for some valuation of NUM not specify.

Verilog compiler put latches(显示之前的value) in the circuit.